I529: Bioinformatics in Molecular Biology and Genetics: Practical Applications **(3 CR)**

HW3 (Due: **March** 25th **BEFORE** Lab session)

http://darwin.informatics.indiana.edu/col/courses/I529-16

INTRODUCTION:

There are three sessions to be completed. The section 1 is for programming using Python or C/C++, the section 2 consists of problems related to computational methods and algorithms and the section 3 is for the group project to be completed by all members in each group. Please submit your completed homework (all sessions, for 1st and 3rd sections, source code should be included along with a report) on the Oncourse. Pdf files are encouraged for session 2; handwritten document scanned in pdf files are accepted, but not preferred for session 2. Each group may submit only one copy of the answer (source code along with a report) for section 3 by one of the group members, and **in the report the responsibility of each group member should be briefly described**.

QUESTION:

Don't hesitate to contact me (Haixu Tang: hatang@indiana.edu).

INSTRUCTION:

1. Please start to work on the homework as soon as possible. For some of you without enough computational background may need much more time than others.
2. Include **README** file for each programming assignment. This is not supposed to be lengthy but should contain concrete and enough information;
   A. Function of the program
   B. Input / Output
   C. Sample usage
3. **Please ENJOY learning and practicing new things.**

WARNINGS: **YOU ARE SUPPOSED TO WORK IN GROUP FOR THE MINI CLASS PROJECT. HOWEVER, YOU MUST DO HOMEWORK SESSION 1 AND 2 ON YOUR OWN.**

---**Section 1** ---

For section 1, you are required to write Python scripts or C/C++ program to do the following tasks.

- Note: Sequence file should be in **FASTA** format. Please refer to the following site for further information on FASTA format; (Reference 1, Reference 2), **40 points.**

A Hidden Markov Model (HMM) is a Markov chain in which the states are not directly observable. Instead, the output of the current state is observable. The output symbol for each state is randomly chosen from a finite output alphabet according to some probability distribution. A *Generalized Hidden Markov Model* (GHMM) generalizes the HMM as follows: in a GHMM, the output of a state may not be a single symbol, but a string of finite length. For a particular hidden state, the length of the output string as well as the output string itself might be chosen according to some probability distributions, which can be different for different states. Formally a GHMM is described by a set of four parameters:

- A finite set Q of hidden states.
- Initial state probability distribution π.
- Transition probabilities $T_{i,j}$ for $i, j \in Q$
- Length distributions f of the states (f_q is the length distribution for state q).
- Probabilistic models for each state, according to which output strings are generated upon visiting a state.

In last assignment, you have suggested a HMM model of the prediction of protein secondary structure using Q3 representation. This time we want to implement the GHHM model to predict protein secondary structure. You can use some parts of code from the group project 2 for this question.

- Procedure (hints)
  - Use one of the protein secondary structure prediction provided protein sequences with known secondary structures as training set (e.g., the benchmark dataset that can be found at https://archive.ics.uci.edu/ml/machine-learning-databases/molecular-biology/protein-secondary-structure/);
  - Build the GHMM for protein secondary structure prediction, and obtain the necessary parameters from the training set; Implement the program (PredProStr_ghmm) based on the Viterbi algorithm to predict the Q3 represented secondary structure for a given protein sequence. Your program should take the same kinds of FASTA format for input and output as “PreProStr_ghmm –i inputfile –o outputfile”.

- Result
  - The program PredProStr_ghmm, including the source code and a short readme file.
  - An example of running your program (input and output).

---**Section 2** --

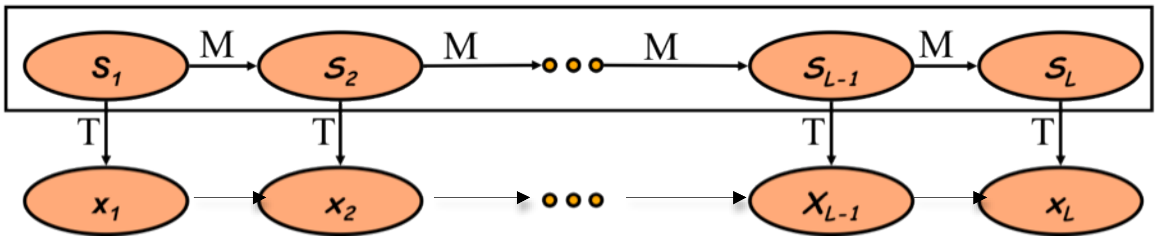
For section 2, you are NOT required to write scripts. **30 points**

1. **Hidden Markov Model Training**. Given sample behavior of HMM, compute the statistical parameters for HMM. Assume we have two coins, one is a fair coin (P(H|F) = 1/2 and P(T|F) = 1/2) and the other is a bias coin (P(H|B) = ? and P(T|B) = ?). Compute the statistical parameters (emission and transition probabilities) for HMM given the sample behavior shown as following:

FFFBFF	BFFBFF	FFBFFF	FFFFBF
HHTHTH	THTHTH	THHTTH	THTTTH
BFFFBF	FFFBBF	BFFFFF	BFBFFF
THHTHT	HHTHHT	HHTTHT	HTTTHH

2. The standard HMM assumes the observations are conditionally independent given the hidden state. In *auto-regressive HMM* (AR-HMM, also known as the regime switching Markov model) has additional dependences from x_{i-1} to x_i, resulting in the modification of the emission probabilities as,

$$p(x_i \mid \theta) = p(x_i \mid x_{i-1}, s_i)$$



AR-HMM essentially combines two Markov chains, one on the hidden variables to capture the long range dependence, and one on the observed variables, to capture short range dependence. Since the observation symbols are known, the connection between them only change the computation of the local evidence; inference can still be performed using the standard Vertebi or forward/backward algorithm. Parameter estimation is also straightforward when both hidden and observation sequences are known, except that the emission probability contains two dependent variables. 1) Describe the Vertebi algorithm for inferring the most likely hidden state sequence from a given observation sequence using the AR-HMM; 2) Describle the methods to estimate the parameters of AR-HMM when the annotated observation sequences are given (i.e. the corresponding hidden sequence of each observation sequence is given in the training set). 3) Compare the number of parameters in the AR-HMM and a regular HMM, assuming the number of hidden states and observation symbols are *m* and *n*, respectively.

3. Given the multiple alignment of several homologous proteins X in multiple species, and the phylogenetic tree (including branch lengths) Ψ among these proteins, one can compute the posterior probability $P(X \mid \psi)$ of observing the sequences X across multiple sites in the alignment (ignoring the sites containing gaps) by using pruning algorithm. 1) Assuming there are two phylogenetic trees Ψ_1 and Ψ_2, which describes the relationship among the segment of these proteins (that are aligned together). Devise a Hidden Markov Model (HMM) to partition the multiple alignment into syntenic blocks between which the phylogenetic relationship is changed from one tree to the other; 2) Assuming we only know one of the two phylogenetic trees Ψ_1, how can we infer the most likely second tree, simultaneously inferring the syntenic blocks.

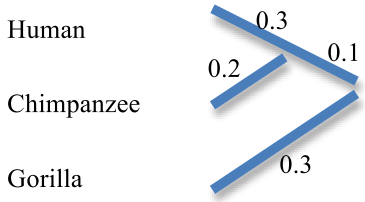
---------------------------------- Mini Group Project # 3 --------------------------------------

(Pruning algorithm) Implement Felsenstein's pruning algorithm on three aligned sequences. (**30 points**)

- Procedure (hints)
  - Input: multiple alignment of three nucleotide sequences with NO GAP, such as

Human:	AGCTTC
Chimpanzee:	AGTTGC
Gorilla:	ACTTGC

  and a phylogenetic tree of these three sequences, following the Newick Standard (.nwk). Newick Standard is a method to describe trees by parentheses and commas. For example, ((Human:0.3, Chimpanzee:0.2):0.1, Gorilla:0.3) represent the following tree:



  - Implement a pruning algorithm to compute the probability to observe these sequences given the phylogenetic tree.
  - Use the base substitution matrix (of the unit evolution time 0.1):

$$P(0.1) = \begin{pmatrix} 0.9 & 0.05 & 0.025 & 0.025 \\ 0.05 & 0.9 & 0.025 & 0.025 \\ 0.025 & 0.025 & 0.9 & 0.5 \\ 0.025 & 0.025 & 0.5 & 0.9 \end{pmatrix}$$

  for the nucleotides {A, C, G, T}.

  - The program should be executed as following:

  pruning tree_file alignment_file

  and the probability will be printed from the screen.

- Result
  - The program, pruning, including the source code and a short readme file.
  - An example of running your program (input and output). You can use the simple example as shown above.